School of Computation, Information and Technology
Technische Universität München

Lehrstuhl für
Rechnerarchitektur & Parallele Systeme
Prof. Dr. Martin Schulz
Dominic Prinz
Jakob Schäffeler

Lehrstuhl für
Design Automation
Prof. Dr.-Ing. Robert Wille
Stefan Engels

Einführung in die Rechnerarchitektur

Wintersemester 2024/2025

Übungsblatt 5: Caching 18.11.2024 – 22.11.2024

1 Speicherzugriffszeit

Gegeben sei ein Programm mit $3 \cdot 10^9$ Instruktionen, das Sie auf einem in-order execution Prozessor mit einer Frequenz von 1.0 GHz ausführen. Dieses Programm läuft auf diesem Prozessor mit durchschnittlich 2 Zyklen pro Instruktion (CPI), exklusive Speicherzugriffe. Jede Instruktion generiere dabei durchschnittlich 1.33 Speicherzugriffe.

a) Wie groß ist die durchschnittliche Speicherzugriffszeit (Average Memory Access Time), wenn das Programm eine Ausführungszeit von 66 Sekunden hat?

$$\text{CPUTIME} = IC \cdot \left(\frac{\text{CPI}}{\text{ins}} + \frac{\text{Memory Accesses}}{\text{Instr}} \cdot \text{AMAT} \right)$$

$$66\,\text{s} = 3 \cdot 10^9\,\text{ins} \cdot \left(\frac{2\ \frac{\text{cycles}}{\text{ins}}}{1 \cdot 10^9\ \frac{\text{cycles}}{\text{s}}} + 1{,}33\ \frac{\text{Mem Accesses}}{\text{Instr}} \cdot \text{AMAT} \right)$$

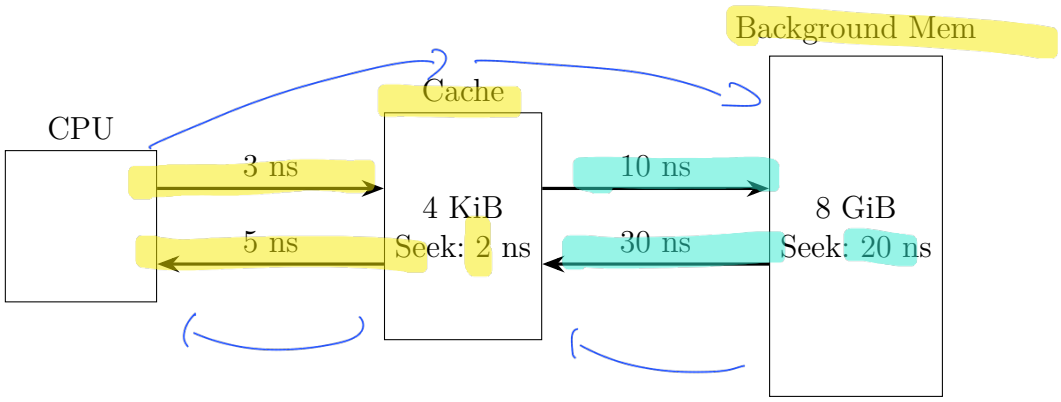
$$66\,\text{s} = \frac{6 \cdot 10^9\ \text{cycles}}{1 \cdot 10^9\ \frac{\text{cycles}}{\text{s}}} + 3 \cdot 10^9\,\text{ins} \cdot 1{,}33\ \frac{\text{Mem Accesses}}{\text{Instr}} \cdot \text{AMAT}$$

$$66\,\text{s} = 6\,\text{s} + 4 \cdot 10^9\ \text{Accesses} \cdot \text{AMAT}$$

$$60\,\text{s} = 4 \cdot 10^9\ \text{Accesses} \cdot \text{AMAT}$$

$$\text{AMAT} = \frac{60\,\text{s}}{4 \cdot 10^9\ \text{accesses}} = 15 \cdot 10^{-9}\ \frac{\text{s}}{\text{access}} = 15\ \frac{\text{ns}}{\text{access}}$$

Der Speicher eines Computers sei wie folgt aufgebaut:



b) Berechnen Sie Hit-Latency, Miss Penalty, Miss-Latency und Miss-Rate.

Hit-Latency: 3 + 2 + 5 = 10 ns

Miss-Lat: 10 + 10 + 20 + 30 = 70 ns

Miss-Penalty: 10 + 20 + 20 = 60 ns

c) Ein*e Programmierer*in optimiert das Programm um die Hit-Rate im Cache zu maximieren. Dadurch senkt sich die durchschnittliche Speicherzugriffszeit um $\frac{1}{3} = 33.33\%$. Dennoch erhöht sich die Ausführungszeit. Woran könnte das liegen?

Folgende Parameter ändern sich verschlechtert:
- Mehr Instr.
- Mehr durchschnittliche Cycles / Instr.
- Mehr Speicherzugriffe / Instr.

d) Angenommen neben der um $\frac{1}{3}$ reduzierten durchschnittlichen Speicherzugriffszeit verändert sich durch die Optimierung nur die Anzahl an Instruktionen. Wie sehr muss sich die Anzahl an Instruktionen relativ verändern, damit sich die Laufzeit nicht verbessert?

Ansatz: Stellen Sie Formel für CPU Time nach IC um und setzen Sie einmal die reguläre Average Memory Access Time und dann die verkürzte Average Memory Access Time ein. Um die relative Abweichung zu berechnen, teilen Sie dann die beiden Instruction Counts durcheinander.

$$AMAT = \text{Hit Rate} \cdot \text{Hit Latency} + \text{Miss Rate} \cdot \text{Miss Latency}$$

$$15\,ns = (1 - \text{Miss Rate}) \cdot 10\,ns + \text{Miss Rate} \cdot 70\,ns$$

$$15\,ns = 10\,ns - \text{Miss Rate} \cdot 10\,ns + \text{Miss Rate} \cdot 70\,ns$$

$$5\,ns = \text{Miss Rate} \cdot 60\,ns$$

$$\text{Miss Rate} = \frac{5\,ns}{60\,ns} = \frac{1}{12} \approx 8{,}3\%$$

---

1) $$IC = \frac{\text{CPU TIME}}{\left(\frac{CPI}{f} + \frac{\text{Num}A}{\text{Inst}_C} \cdot AMAT\right)}$$

$$\frac{IC_2}{IC_1} = \frac{\dfrac{\text{CPU TIME}}{\left(\frac{CPI}{f} + \frac{\text{Num}A}{\text{Inst}_C} \cdot AMAT_2\right)}}{\dfrac{\text{CPU TIME}}{\left(\frac{CPI}{f} + \frac{\text{Num}A}{\text{Inst}_C} \cdot AMAT_1\right)}}$$

$$\frac{IC_2}{IC_1} = \frac{\left(\frac{CPI}{f} + \frac{\text{Num}A}{\text{Inst}_C} \cdot AMAT_1\right)}{\left(\frac{CPI}{f} + \frac{\text{Num}A}{\text{Inst}_C} \cdot AMAT_2\right)}$$

$$= \frac{\left(\frac{2\ \frac{cycles}{ins}}{10^9\ \frac{cycles}{s}} + 1{,}33\ \frac{accesses}{ins} \cdot 15\ \frac{ns}{access}\right)}{\left(\frac{2\ \frac{cycles}{ins}}{10^9\ \frac{cycles}{s}} + 1{,}33\ \frac{accesses}{ins} \cdot \left(15 - \frac{1}{3} \cdot 15\right)\frac{ns}{access}\right)}$$

$$= \frac{\left(2 \cdot 10^{-9}\ \frac{s}{ins} + 1{,}33\ \frac{accesses}{ins} \cdot 15 \cdot 10^{-9}\ \frac{s}{access}\right)}{\left(2 \cdot 10^{-9}\ \frac{s}{ins} + 1{,}33\ \frac{accesses}{ins} \cdot 10 \cdot 10^{-9}\ \frac{s}{access}\right)}$$

$$= \frac{\left(2\ \frac{s}{ins} + 1{,}33\ \frac{accesses}{ins} \cdot 15 \cdot \frac{s}{access}\right)}{\left(2\ \frac{s}{ins} + 1{,}33\ \frac{accesses}{ins} \cdot 10 \cdot \frac{s}{access}\right)} = \frac{21{,}95\ \frac{s}{ins}}{15{,}3\ \frac{s}{ins}}$$

$$= 1{,}434$$

e) Was sind Gründe, weshalb sich die Laufzeiten eines Programms heutzutage nicht mehr so einfach wie oben berechnen lassen?

- Parallelisierung : mehrere Programme (z.B. OS)
- Verschiedene Cache-Ebenen (werden zwischen Prozessoren geteilt)
- Moderne Prozessoren suchen Info um memory stalls zu vermeiden

2 Vergleich der Cachearten

Gegeben sei eine 16-Bit Architektur mit byte-adressierbarem Speicher. Zudem ist ein 256 Byte großer Cache mit 2 Byte langen Cachezeilen gegeben. Vergleichen Sie in den folgenden Teilaufgaben die verschiedenen Cache-Typen, indem Sie die Fragen jeweils für einen direct mapped Cache, einen 4-fach assoziativen Cache und einen voll-assoziativen Cache beantworten.

a) In wie viele unterschiedliche Cachezeilen kann ein Byte aus dem Speicher potentiell gecached werden?

256 Byte / 2 Byte = 128 Cachelines

Voll-assoz: 128
4-fach-assoz: 4
Direct mapped: 1

b) Es wird auf das Byte an der Adresse 0xABCD zugegriffen. Wie viele unterschiedliche Cachezeilen müssen überprüft werden, um festzustellen, ob das Datum an der Adresse bereits gecached ist? Geben Sie zudem an, welcher Vergleich stattfindet, um festzustellen, ob der Wert bereits gecached ist.

Tag: 0b 1010 1011 1100 1101

0xABCD = 0b 1010 1011 1100 1101

Voll-assoz: Offset-Bits: 1 Index: 0 Tag Bits: 16 − 1 = 15

4-fach-assoz: Offset-Bits: 1 $\frac{128}{4} = 32$ Cachesets Index: 5 Tag Bits: 16 − 5 − 1 = 10
Tag: 0b 1010 1011 11

Direct mapped: Offset-Bits: 1 Index: 7 Tag Bits: 16 − 7 − 1 = 8
Tag: 0b 1010 1011

3 Cachestruktur und Cachemisses

Für die folgenden Fragen wird ein System betrachtet, das nur einen Rechenkern mit einem Cache besitzt. Dieser Cache sei 2-fach assoziativ, habe 4 Cachezeilenmengen (Cache Sets) und eine Cachezeilenlänge von 32 Byte. Die Architektur sei eine reine 32-Bit Architektur und alle Zugriffe 32 Bit breit. Der Speicher sei wie üblich Byte-adressierbar.

a) Wie viele Cachezeilen hat der Cache?

4 Cachesets * 2 $\frac{\text{Cachelines}}{\text{Cacheset}}$ = 8 Cachelines

b) Wie viele Bits benötigt man für Offset, Index und Tag?

Offset: $\lceil \log_2 32 \rceil$ = 5 Bits
Index: $\lceil \log_2 4 \rceil$ = 2 Bits
Tag: 32 − 5 − 2 = 25 Bits

c) Gegeben sei folgender Ausschnitt eines C-Programms:

```
int data[1000];
int sum = 0;

// ABSCHNITT 1
for (int i = 0; i < 1000; i += 64) {
    sum = sum + data[i];
}

// ABSCHNITT 2
for (int i = 0; i < 1000; i += 32) {
    sum = sum + data[i];
}
```

Anzahl Zugriffe: $\lceil \frac{1000}{64} \rceil = 16$

0, 64, 128, ...

Anzahl Zugriffe: $\lceil \frac{1000}{32} \rceil = 32$

0, 32, ~~64~~, 96, ~~128~~ ...

Nehmen Sie an, dass nur das Array im Speicher liegt und alle anderen Werte in Registern gehalten werden. Der Cache sei zu Beginn leer. Gehen Sie davon aus, dass ein `int` 32-Bit groß ist und stets vollständig in einer Cachezeile liegt.

i) Wie viele „Capacity Misses“ sind in den beiden Abschnitten zu erwarten?

AD 1: $64 \cdot 4 = 256$

AD 2: $32 \cdot 4 = 128$

Iteration	Tag	Index	Offset
1	0000000000	00	00000
2	0000000001	00	00000
3	0000000010	00	00000
4	0000000011	00	00000

⇒ immer auf Cache-Set 0 abbilden

Abschnitt 1: 0 Capacity Misses

Abschnitt 2: 16 CM

ii) Wie viele „Cold Misses“ sind zu erwarten?

Ansatz: Betrachten Sie die Adressen der einzelnen Array-Elemente.

Abschnitt 1: 16 Cold Misses

Abschnitt 2: 16 Cold Misses

iii) Knobelaufgabe: Wie ändern sich diese Zahlen, wenn die Variablen `sum` und `i` nicht in Registern sondern direkt nach dem Array `data` befinden?

4 Ersetzungsstategien

Gegeben sei ein 4-fach assoziativer Cache mit 16 Cachezeilen. Pro Cachezeile werden 4 Byte abgespeichert. In den folgenden Tabellen ist jeweils das Cache Set 0 angegeben. Cachen sie die folgenden Werte jeweils einmal mit LFU und LRU als Ersetzungsstategie und füllen sie in den Tabellen aus. Als Vereinfachung wird von jeder Adresse nur das Tag angegeben und Index und Offset werden in dieser Aufgabe ignoriert. Falls die Ersetzungsstrategie zu keinem eindeutigen Ergebnis führt ersetzen Sie immer die niederwertigere Cacheline mit dem neuen Tag.

a) Tragen Sie in die Tabelle die für die Blöcke gespeicherten Tags mit LRU als Ersetzungsstategie ein.

*Beispiel: In Zeitschritt 1 wird die Adresse mit dem Tag 12 gecached. Da das Datum an der Adresse noch nicht im Cache vorhanden ist handelt es sich um ein Miss. Der Wert wird in Cachezeile 1 und der letzte Access in **L**ast **U**se von Cachezeile 1 gespeichert.*

Time	Tag	Line 0		Line 1		Line 2		Line 3		Hit/Miss
		Tag	LU	Tag	LU	Tag	LU	Tag	LU	
1	12	12	1	–	–	–	–	–	–	Miss
2	20	12	1	20	2					Miss
3	12	12	3	20	2					Hit
4	12	12	4	20	2					Hit
5	28	12	4	20	2	28	5			Miss
6	32	12	4	20	2	28	5	32	6	Miss
7	38	12	4	38	7	28	5	32	6	Miss
8	25	25	8	38	7	28	5	32	6	Miss
9	12	25	8	38	7	12	9	32	6	Miss
10	32	25	8	38	7	12	9	32	10	Hit

b) Tragen Sie in die Tabelle die für die Blöcke gespeicherten Tags mit LFU als Ersetzungsstategie ein.

Beispiel: In Zeitschritt 1 wird die Adresse mit dem Tag 12 gecached. Da das Datum an der Adresse noch nicht im Cache vorhanden ist handelt es sich um ein Miss. Der Wert wird in Cachezeile 1 gespeichert und der Counter auf 1 gesetzt.

Time	Tag	Line 0		Line 1		Line 2		Line 3		Hit/Miss
		Tag	Count	Tag	Count	Tag	Count	Tag	Count	
1	12	12	1	–	–	–	–	–	–	Miss
2	20	12	1	20	1					Miss
3	12	12	2	20	1					Hit
4	12	12	3	20	1					Hit
5	28	12	3	20	1	28	1			Miss
6	32	12	3	20	1	28	1	32	1	Miss
7	38	12	3	38	1	28	1	32	1	Miss
8	25	12	3	29	1	28	1	32	1	Hit
9	12	12	4	29	1	28	1	32	2	Hit
10	32	12	4	29	1	28	1			

c) Vergleichen Sie die Ersetzungsstategien. Was sind Vor- und Nachteile der jeweiligen Strategien?

LFU: Gut bei repetitiven Zugriffsmustern, einfach zu implementieren
schlecht bei unregelmäßigen Zugriffsmustern

LRU: Schwierig zu implementieren; schlecht bei zufälligen oder wechselnden Zugriffs-

5 Cache-Simulation (Hausaufgabe 5)

Bearbeitung und Abgabe der Hausaufgabe 5 auf `https://artemis.in.tum.de/courses/401` bis **Sonntag, den 24.11.2024, 23:59 Uhr**.